

# Lecture 1.5 :git/LaTeX

## Fastcampus Math Camp

신승우

Sunday 27<sup>th</sup> May, 2018

# Outline

- 1 Introduction to Git
- 2 Introduction to LaTeX

# Outline

- 1 Introduction to Git
- 2 Introduction to LaTeX

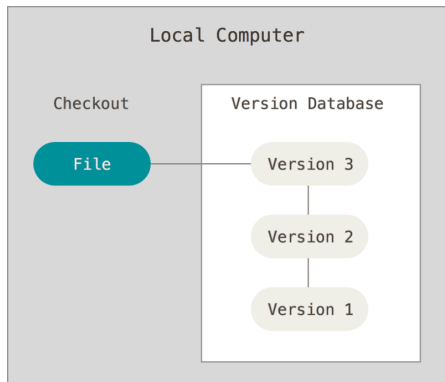
$$\sum_{i=1}^n$$

- Windows : git-scm 설치 (<http://git-scm.com/download/win>)
- Linux(Ubuntu) : `sudo apt-get install git`
- Mac : git-scm 설치 (<http://git-scm.com/download/mac>)

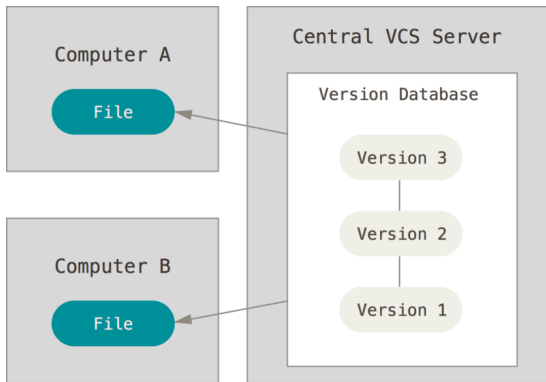
# Git 이란?

- 버전 관리 툴(VCS: Version Control System)
- Made by 'The' Linus Benedict Torvalds

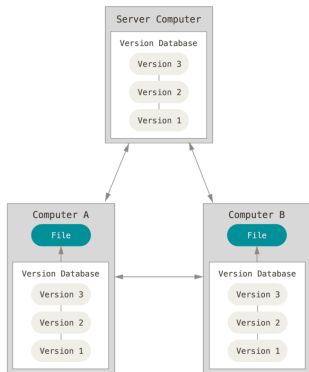
# Local VCS



# Centralized VCS



# Distributed VCS





- 사용자 정보 설정

`git config --global user.name "John Doe"`

`git config --global user.email johndoe@example.com`

- 편집기 설정

`git config --global core.editor emacs (emacs user)`

`git config --global core.editor "path to editor exe"`

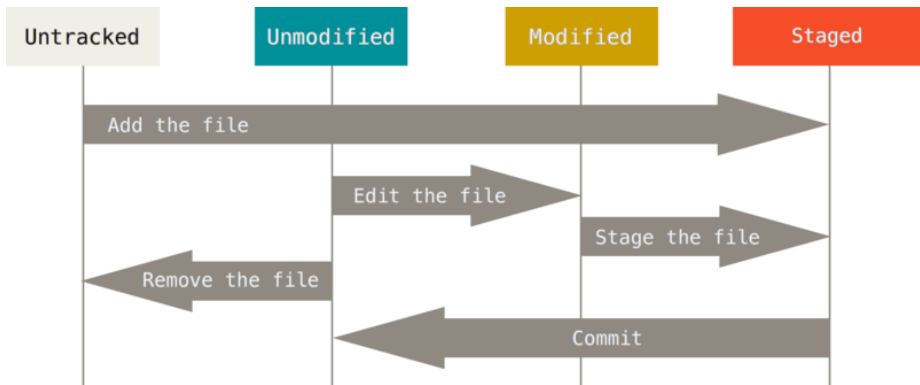
# Git 저장소 만들기

- git init : 새로 만들기
- git clone : 기존 저장소 복사

# Git 안에서 파일의 life cycle 이해

깃은 Snapshot을 저장하는데, 여기서 스냅샷을 '찍는' 행위를 커밋이라고 합니다. 깃은 커밋 시마다 모든 파일의 복사, 압축본을 만들어 저장합니다.

# Git 안에서 파일의 life cycle 이해



# Git 안에서 파일의 life cycle 이해

- Tracked/Untracked : 추적 여부. git add로 추적 시작, git remove로 추적 종료
- Tracked Files
  - Staged : commit 직전
  - Modified : commit 후 수정됨
  - Unmodified : commit 직후

# Git 안에서 파일의 life cycle 이해 : .gitignore

특정 파일은 항상 무시하고 싶을 때, 매번 일일이 add를 할 수는 없다.  
따라서 보통 `git add .` 으로 모든 파일을 add하게 되는 경우가 많은데, 이  
때 원하지 않는 형식의 파일은 add하면 안 된다.

- 확장자 : \*.py 형식
- 무시하는 것에서 예외 : !test.py
- 현 디렉토리 파일만 무시 : /TODO
- 특정 디렉토리 무시 : build/
- 특정 디렉토리 아래의 특정 확장자 무시 : doc/\*\*/\*.\*py
- 주석 : #

# Git에서 수정 후 커밋해보기

일반적으로 다음의 과정을 따른다.

- git init/git clone으로 저장소 시작
- 파일 수정
- git add filename (혹은 git add .)
- git commit -m 'commit message' : 커밋 메세지 없이는 커밋되지 않으며, -m 플래그를 쓰지 않으면 자동으로 위에서 설정한 에디터가 켜집니다.
- git pull origin master : 협업자와 같이 remote에서 작업할 경우
- git push origin master : remote에서 작업할 경우

# Git 에서 전 상태로 되돌리기 : 커밋 수정하기

```
git commit -amend
```

로 마지막 커밋을 수정한다. 커밋 메시지를 바꾸거나, 파일을 빠트렸거나 할 때 사용한다. 이 때, 새로운 커밋은 마지막 커밋을 덮어쓰게 된다. 예를 들어, 다음의 명령어는 하나의 커밋만을 만든다.

```
git commit -m 'hello!'
```

```
git add forgottenfile
```

```
git commit -amend
```



git remote로 remote 저장소를 관리한다.

- git remote : 리모트 저장소의 목록을 보여준다. -v를 쓸 경우, url까지 보여준다.
- git remote add : 리모트 저장소를 추가한다.  
git remote add origin 'https://github.com/yourid/repourl'

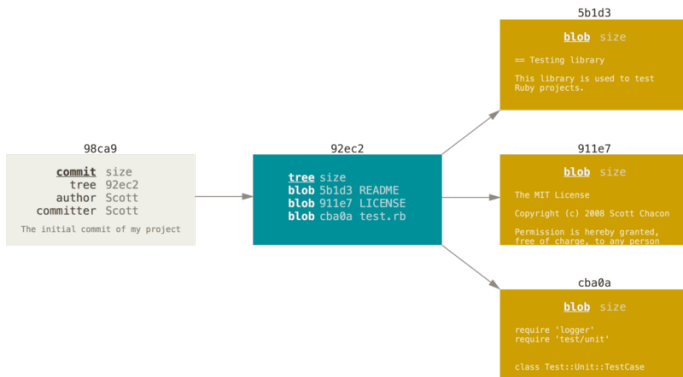
# Remote에 push하기

`git push remote-name branch-name`

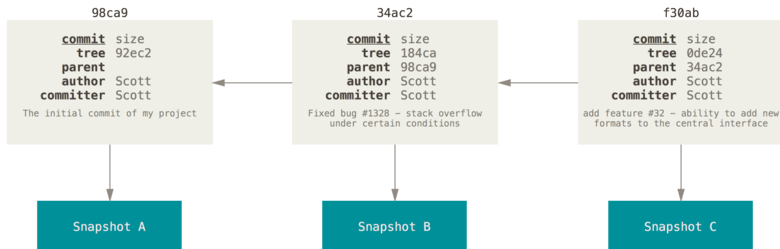
으로 리모트 저장소로 나의 작업을 push할 수 있다. 다만, 여기서 두 가지 조건이 필요한데

- 쓰기 권한이 있어야 함
- 다른 사람의 작업본을 로컬에서 최신 버전으로 업데이트했어야 함

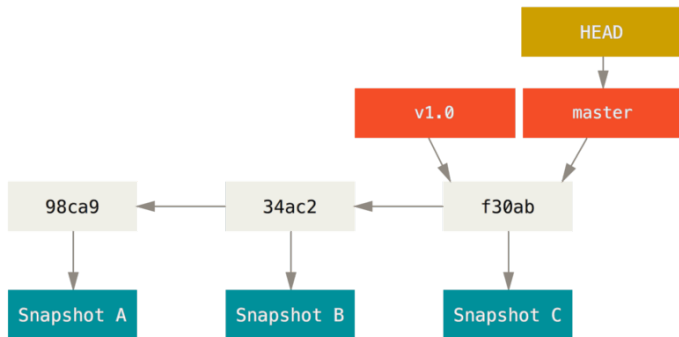
# More on Git Structure - commit internal



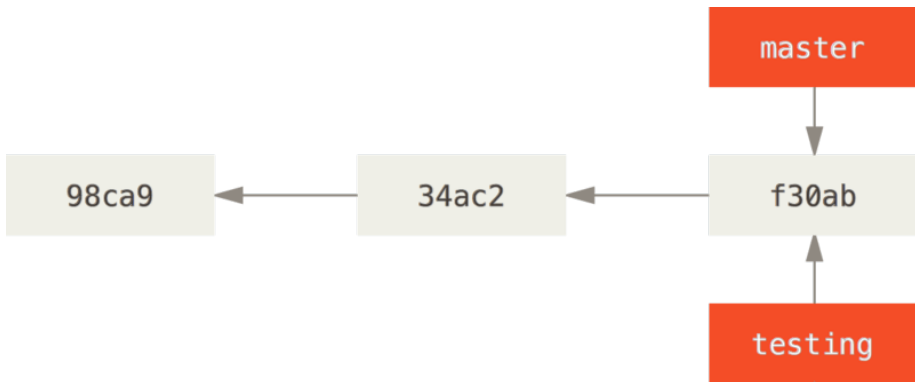
# More on Git Structure - commits



# More on Git Structure - branch

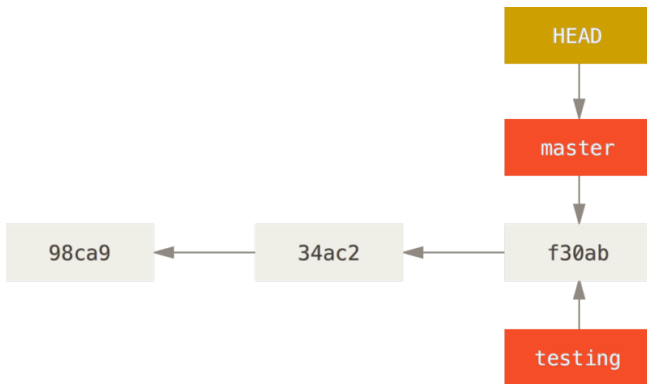


## More on Git Structure - new branch



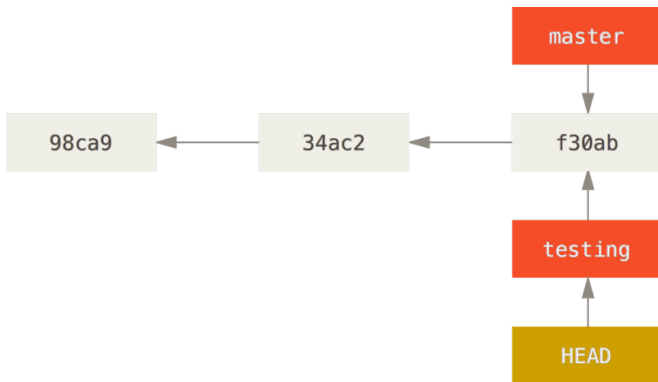
git branch testing으로 새 브랜치를 생성

# More on Git Structure - HEAD



HEAD는 특수한 포인터로, 지금 작업 중인 branch를 가리킴.

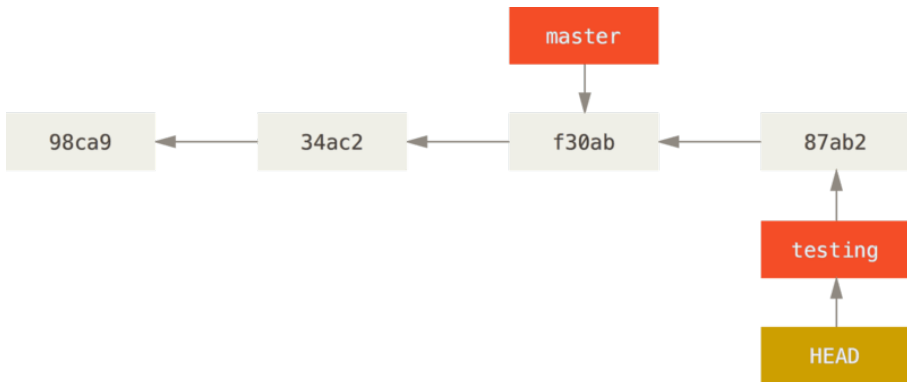
## More on Git Structure - move HEAD



git checkout 명령어로 작업 중인 branch를 바꿀 수 있음. 이 때 HEAD가 이동함.

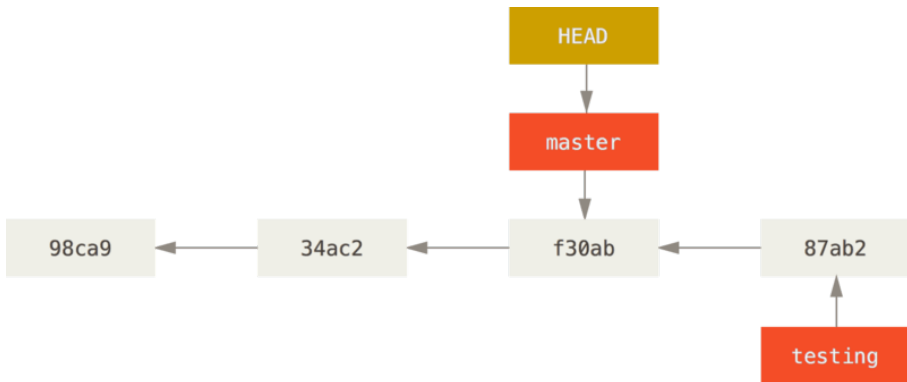


## More on Git Structure - new commit



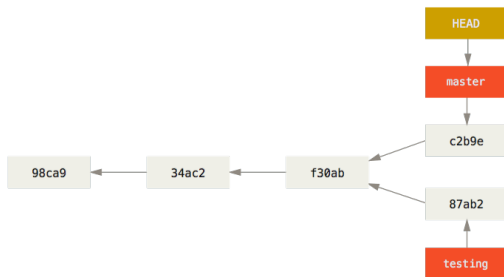
이후, 수정하고 커밋을 해 보면 위와 같다.

# More on Git Structure - new branch



git branch testing으로 새 브랜치를 생성하면 위와 같이 됨.

# More on Git Structure

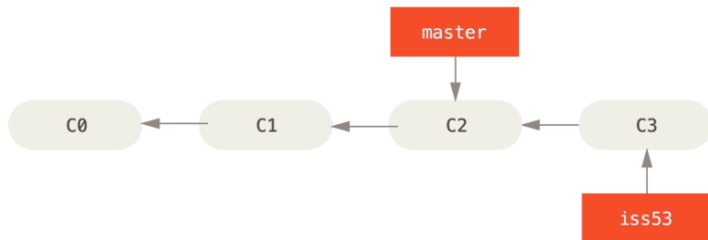


master branch를 수정하여 커밋하였으므로 그림처럼 branch가 나뉘지게 됨.

# Git Branch를 이용한 문제 해결 : Hotfix Issue

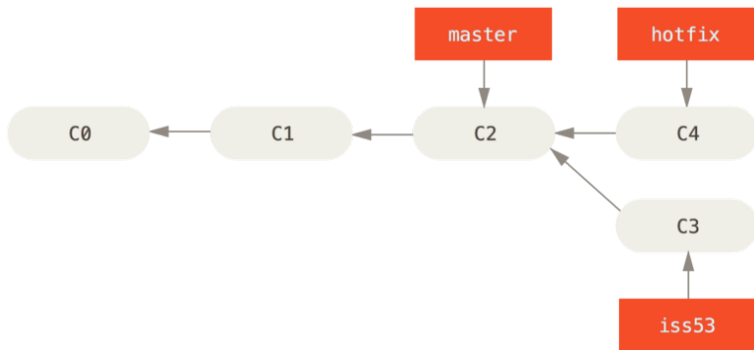
웹사이트 개발 프로젝트를 진행하는 중, 새로운 기능을 넣고 있습니다. 이 때 급하게 고쳐야 하는 hotfix가 있다고 합니다. 기존 진행중이던 기능 개발을 최대한 영향 주지 않으면서 일단 hotfix를 진행하고자 합니다. 이 때 어떤 식으로 버전관리를 해야 할까요?

# Git Branch를 이용한 문제 해결 : Hotfix Issue



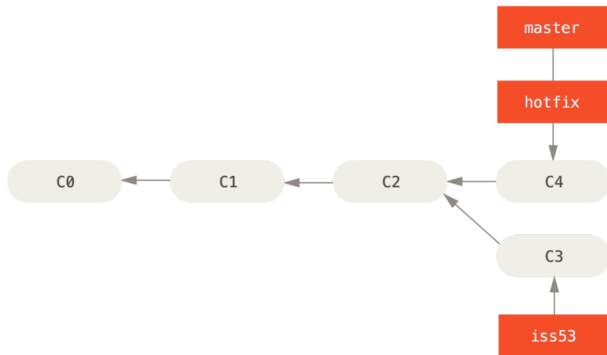
기능 개발 부분이 iss53 브랜치라고 하고, 지금 진행중인 상태를 나타내면 위와 같음.

# Git Branch를 이용한 문제 해결 : Hotfix Issue



이 때, hotfix를 위해서 새로운 branch hotfix를 하나 만듬. // `git checkout -b hotfix`

# Git Branch를 이용한 문제 해결 : Hotfix Issue

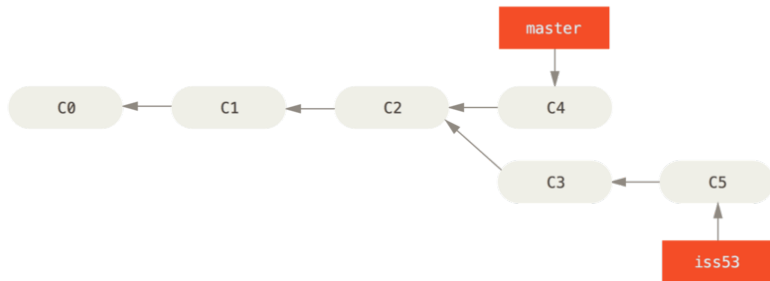


이후 hotfix가 진행되어서, 문제가 해결되면 master를 hotfix와 merge git checkout master

git merge hotfix

이런 형태의 merging을 fast-forward라고 합니다. (직계 조상인 경우)

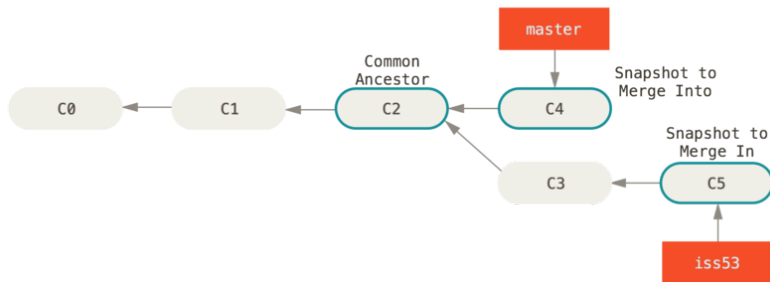
# Git Branch를 이용한 문제 해결 : Hotfix Issue



이 때, 동시에 iss53은 독립적으로 진행 가능



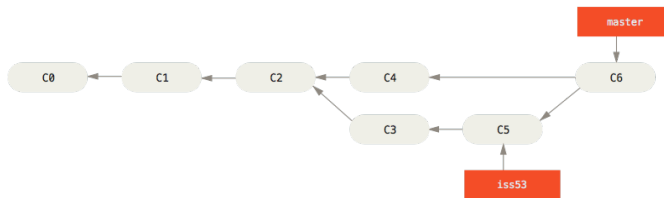
# Git Branch를 이용한 문제 해결 : Hotfix Issue



이제 iss53이 잘 개발이 끝났으므로, master branch와 merge 시도. 이 경우, 공동 조상이 없으므로 가장 가까운 common ancestor를 찾아 merge를 시도한다.

`git merge iss53`

# Git Branch를 이용한 문제 해결 : Hotfix Issue



해결이 완료되면 위와 같이 된다.

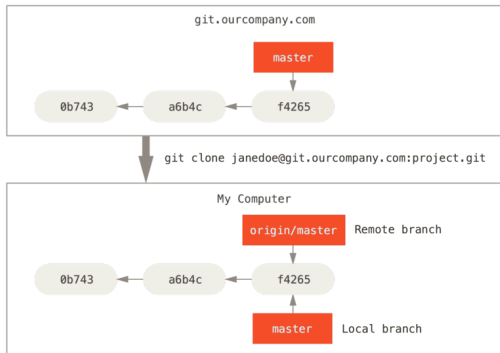
# Merge Conflict

만약, 두 커밋에서 같은 파일의 같은 부분을 다르게 수정했다면 문제가 생긴다. 이런 경우를 merge conflict라고 한다. mergetool을 쓰거나, 수동으로 merge할 수 있다.

## More on Branching : Remote Branch

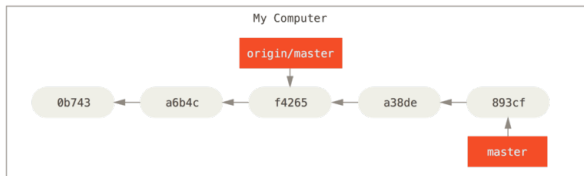
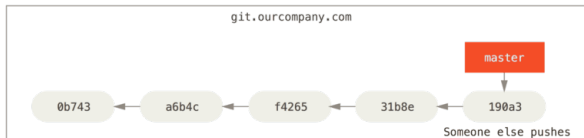
이때까지는 local에서 혼자 작업하는 경우를 고려하였다. 이제, remote에서 협업할 때 어떤 식으로 branching이 사용되는지 살펴보겠다.

# More on Branching : Remote Branch



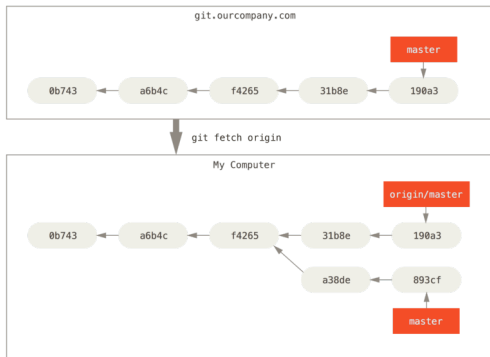
git clone을 사용하여 remote branch를 복사해온다.

# More on Branching : Remote Branch



그 후, remote와 local에서 독립적으로 작업이 진행되었다고 하자.

# More on Branching : Remote Branch



이 때, fetch나 pull을 이용하여 작업된 부분을 받아올 수 있다.

# More on Branching : fetch vs pull

fetch와 pull은 둘 다 리모트 서버에서 데이터를 받아오는 방법이다.  
하지만, 그 후 자동으로 merge를 수행하는지에 대한 차이가 있다. 즉,

```
git fetch origin master
```

```
git merge origin/master
```

과

```
git pull origin master
```

은 같은 작업이다. 일반적으로는 전자가 권장된다.



# Outline

- 1 Introduction to Git
- 2 Introduction to LaTeX

- 로컬 : <https://www.latex-project.org/get/> 에서 해당되는 OS로 설치
- 웹
  - overleaf : <https://www.overleaf.com/>
  - sharelatex : <https://www.sharelatex.com/project>

- 조판(Typesetting) 시스템: 스타일과 내용의 분리!
- Made by 'The' Knuth

- html과 매우 비슷함
- `\`으로 명령어 시작 : `\command{argument}`
- `\begin{environment}`로 특정 환경 시작. ex) `lstlisting`, `equation` 등
- `chapter`, `section`, `subsection`으로 문서 구성

`\usepackage{packagename}`으로 사용. 대표적인 패키지들은

- kotex : 한국어 사용
- lstlisting : 소스 코드 임베딩
- graphicsx : 그래픽 관련. 그림 그리거나 등등
- hyperref : 하이퍼링크용
- lipsum : 템플릿용 dummy text

템플릿을 이용하는 법은 상황 따라 다른데,

- .cls파일 제공 : documentclass
- .sty파일 제공 : usepackage

LaTeX에서는 강력한 수식 입력 툴을 제공하는데, 기본적으로는 hwp 수식입력기와 비슷하다.

- \$\$ 환경
- equation 환경

- 논문 조판
- 교재 조판
- ppt 조판



- detexify : <http://detexify.kirelabs.org/>
- latex table generator : <https://www.tablesgenerator.com/>